


(Optional) JWT Token Authentication

JSON Web Token for Secure API Access

- JWT (JSON Web Token)
 -  JWT vs Session
 - What is JWT?
 - SimpleJWT
- Example
 - test.php
 - index.html
 - protected.php
 - login.php


JWT (JSON Web Token)

Imagine you go to a candy store 🍭.








- You show your special sticker to the shopkeeper, and they say, "Okay! You're allowed to get candy!" 🎉

Then, you go to the toy store 🧸, and you show the same sticker.

- They say, "Okay! You're allowed to get toys too!" 🎉

- That sticker is like JWT — a little tag that says “The person with the sticker is allowed!”
 - You don’t have to tell your name every time.
Just show the sticker! 
- In web programming, we use JWT to help computers know who you are and what you’re allowed to do, without asking again and again.

JWT vs Session

Feature	JWT (Token)	Session (ID)
 Storage	Client (browser)	Server
 Scalable	 Yes (stateless)	 Needs sync/store
 Persist	Until expired	Until logout/expire
 Security	Must sign + verify	Hidden on server
 Use Case	APIs, mobile apps	Web login systems

 **JWT:** Great for APIs & scaling

 **Session:** Simple & secure for websites

What is JWT?

- **JSON Web Token** - A secure way to transmit information
- Contains 3 parts: **Header.Payload.Signature**
- Used for **API authentication** and **user sessions**
- **Stateless** - No server-side storage needed

Why Use JWT?

- ✓ **Stateless** - No database queries needed
- ✓ **Portable** - Works across different services
- ✓ **Secure** - Cryptographically signed
- ✓ **Standard** - RFC 7519 specification

JWT Structure

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImpvaG4iLCJleHAiOjE2ODc1NjAwMDB9.  
abc123signature
```

Header | Payload | Signature

- Algorithm | User data | Verification
- Token type | Claims | Secret key

Creating JWT Token

```
// Generate token
$payload = [
    'user_id' => 123,
    'username' => 'john',
    'exp' => time() + 3600 // 1 hour
];

$token = jwt_encode($payload, $secret_key);
```

Verifying JWT Token

```
// Verify token
try {
    $decoded = jwt_decode($token, $secret_key);
    $user_id = $decoded['user_id'];
    echo "Welcome " . $decoded['username'];
} catch (Exception $e) {
    echo "Invalid token!";
}
```

SimpleJWT

- We make a simple class for JWT token generation.
 - DO NOT use in production - use firebase/php-jwt instead
- It has an encode/decode function to generate a JWT token using HMAC and BASE64 encoding/decoding.

```

<?php
/**
 * Simple JWT Implementation for Educational Purposes
 */
class SimpleJWT {

    public static function encode($payload, $secret) {
        // Create header
        $header = json_encode(['typ' => 'JWT', 'alg' => 'HS256']);

        // Encode header and payload
        $headerEncoded = self::base64UrlEncode($header);
        $payloadEncoded = self::base64UrlEncode(json_encode($payload));

        // Create signature
        $signature = hash_hmac('sha256', $headerEncoded . "." . $payloadEncoded, $secret, true);
        $signatureEncoded = self::base64UrlEncode($signature);

        // Return JWT token
        return $headerEncoded . '.' . $payloadEncoded . '.' . $signatureEncoded;
    }

    private static function base64UrlEncode($data) {
        return rtrim(strtr(base64_encode($data), '+/', '-_'), '=');
    }
}

```

```

public static function decode($jwt, $secret) {
    // Split the JWT
    $parts = explode('.', $jwt);

    if (count($parts) != 3) { throw new Exception('Invalid JWT format'); }

    list($headerEncoded, $payloadEncoded, $signatureEncoded) = $parts;

    // Verify signature
    $expectedSignature = hash_hmac('sha256', $headerEncoded . "." . $payloadEncoded, $secret, true);
    $providedSignature = self::base64UrlDecode($signatureEncoded);

    if (!hash_equals($expectedSignature, $providedSignature)) {
        throw new Exception('Invalid signature');
    }
    // Decode payload
    $payload = json_decode(self::base64UrlDecode($payloadEncoded), true);
    // Check expiration
    if (isset($payload['exp']) && $payload['exp'] < time()) {
        throw new Exception('Token has expired');
    }
    return $payload;
}

private static function base64UrlDecode($data) {
    return base64_decode(str_pad(strtr($data, '-_', '+/'), strlen($data) % 4, '=', STR_PAD_RIGHT));
}
}
?>

```

Example

- We use the SimpleJWT class to see how a JWT token is used.
- test.php
 - create, decode, show, and test expired/invalid token
- index.html
 - login.php to get JWT Token
 - protected.php to use the JWT Token

test.php

- Run this script on the web browser to see JWT encoding/decoding in action.
 - It uses SimpleJWT.php
- Run `curl http://localhost:8000/test.php` to see the results.

Step 1: Create token

```
$payload = [  
    'user_id' => 123,  
    'username' => 'john',  
    'role' => 'user',  
    'iat' => time(),  
    'exp' => time() + 3600 // 1 hour  
];  
$token = SimpleJWT::encode($payload, JWT_SECRET);
```

Step 2: Decode the token

```
$decoded = SimpleJWT::decode($token, JWT_SECRET);
```

Step 3: Show token structure

```
$parts = explode('.', $token);  
$header = json_decode(base64_decode(str_pad(strtr($parts[0], '-_', '+/'), strlen($parts[0]) % 4, '=', STR_PAD_RIGHT)), true);
```


Step 4: Test expired token

```
$expiredPayload = [  
    'user_id' => 123,  
    'username' => 'john',  
    'exp' => time() - 3600 // Already expired  
];  
  
$expiredToken = SimpleJWT::encode($expiredPayload, JWT_SECRET);  
try {  
    SimpleJWT::decode($expiredToken, JWT_SECRET);  
    echo "✗ This should not happen – expired token was accepted!\n";  
} catch (Exception $e) {  
    echo "✓ Expired token correctly rejected: " . $e->getMessage() . "\n\n";  
}
```

Step 5: Test invalid signature

```
$tamperedToken = $token . 'tampered';  
try {  
    SimpleJWT::decode($tamperedToken, JWT_SECRET);  
    echo "❌ This should not happen – tampered token was accepted!\n";  
} catch (Exception $e) {  
    echo "✅ Tampered token correctly rejected: " . $e->getMessage() . "\n\n";  
}
```

index.html

Let's build a simple JWT authentication system!

```
// Login → Generate Token → Access Protected Route
```

Step 1. Get JWT Token

```
// In index.html - login() function
const response = await fetch('login.php', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ username, password })
});
```

Step 2. Store the Token

Then, the token is automatically stored.

```
// Token is stored in the input field  
document.getElementById('token').value = data.token;
```

Step 3. Access Protected Resource

```
// This is the key part – sending Bearer token!
const response = await fetch('protected.php', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${token}` // 🔑 This is how you access protected.php!
  }
});
```

protected.php

1. Receives the token

```
GET /protected.php HTTP/1.1  
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyYXkiOiJmcm9udCIsImF1dG8iOiJlbnRydWUiLCJpc3MiOiJkaXIifQ.ywE6Mg (many more) wfQ.signature...
```

2. Processes the token

```
// Extract Bearer token
preg_match('/Bearer\s+(.*)$/i', $authHeader, $matches);
$token = $matches[1];

// Validate JWT
$payload = SimpleJWT::decode($token, JWT_SECRET);
```

3. Return protected data if valid

```
sendJsonResponse([
    'message' => 'Access granted to protected resource',
    'user_data' => [
        'user_id' => $payload['user_id'],
        'username' => $payload['username'],
        'role' => $payload['role']
    ],
    'token_info' => [
        'issued_at' => date('Y-m-d H:i:s', $payload['iat']),
        'expires_at' => date('Y-m-d H:i:s', $payload['exp']),
        'time_remaining' => $payload['exp'] - time() . ' seconds'
    ],
    'protected_data' => [
        'secret_message' => 'This is confidential information!',
        'server_time' => date('Y-m-d H:i:s'),
        'access_level' => $payload['role']
    ]
]);
```


login.php

- Create JWT payload

```
$payload = [
    'user_id' => $user['id'], 'username' => $user['username'],
    'role' => $user['role'], 'iat' => time(),                // Issued at
    'exp' => time() + 3600                                // Expires in 1 hour
];
try {
    // Generate token
    $token = SimpleJWT::encode($payload, JWT_SECRET);
    sendJsonResponse([
        'message' => 'Login successful',
        'token' => $token,
        'expires_in' => 3600,
        'user' => [
            'id' => $user['id'], 'username' => $user['username'], 'role' => $user['role']
        ]
    ]);
} catch (Exception $e) {
    sendJsonResponse(['error' => 'Token generation failed'], 500);
}
```